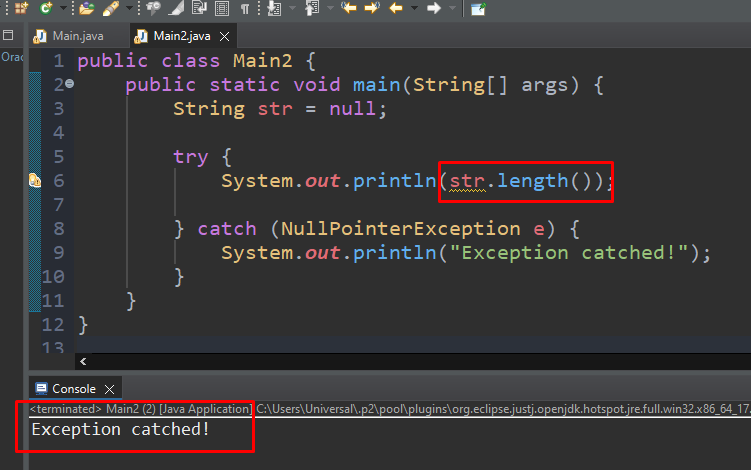
Javada Exception larni 3 xil turi bor:

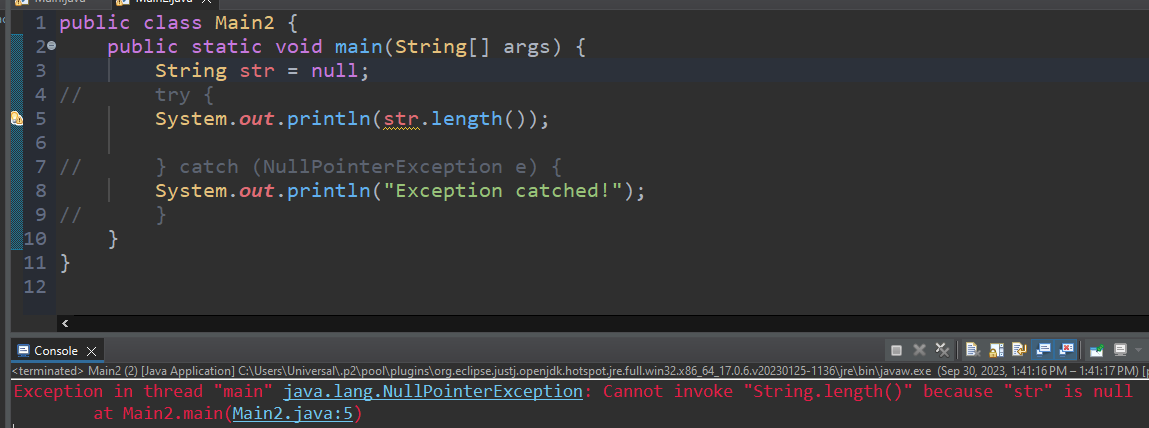


Keling avval unchecked exception bilan tanishaylik. Unchecked exceptionni biz Runtime exception deb ham ataymiz. Chunki bu turdagi exceptionlar kodimizni run qilgandagina tashlanadi. **Shuning uchun bu turdagi exceptionlarda kodimizni try catchga o’rash shart emas, o’rasak ham bo’ladi lekin tavsiya etilmaydi. Lekin checked exceptionlarni albatta try-catchga o’rash shart.**

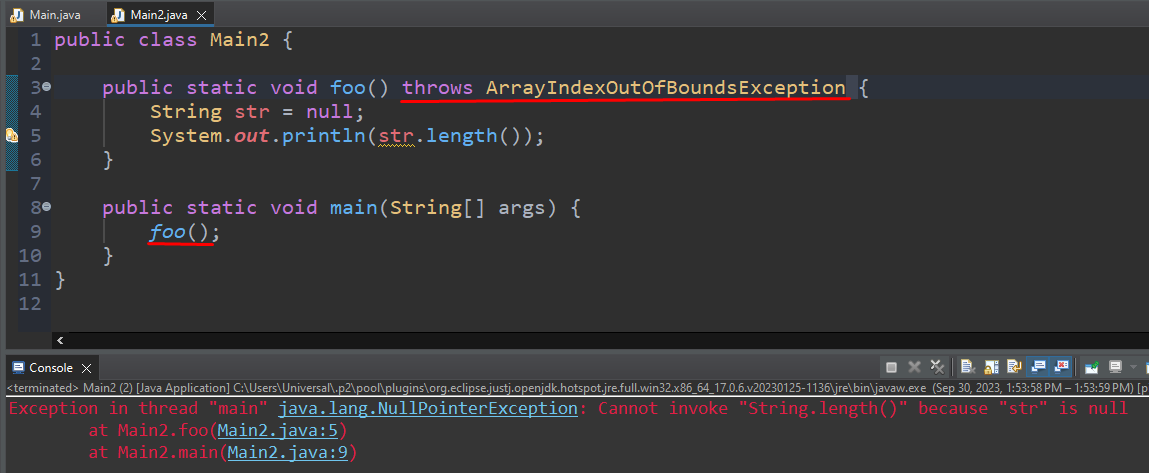
Masalan, pastdagi kodimizda 6-qatordagi kodni sout(str.length()); qismida xatolik chiqadi, buni bilamiz chunki str=null dir. Lekin bu xatolikni compile timeda emas, balki runtime da olamiz. Shuning uchun try-catch ni yozish ixtiyoriydir:



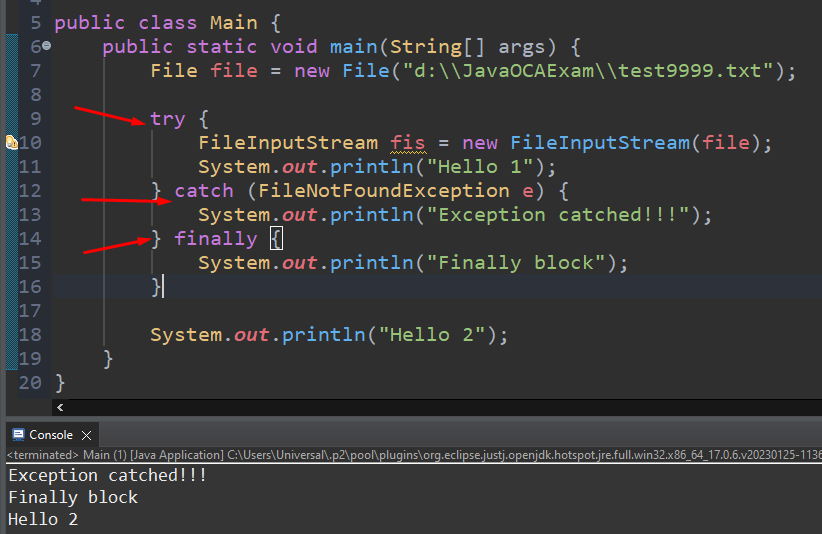
Try-catch ni yozmasak ham compile time da xatolik bermaydi. Faqat compile bo’lgach xato tashlanadi console da xolos:



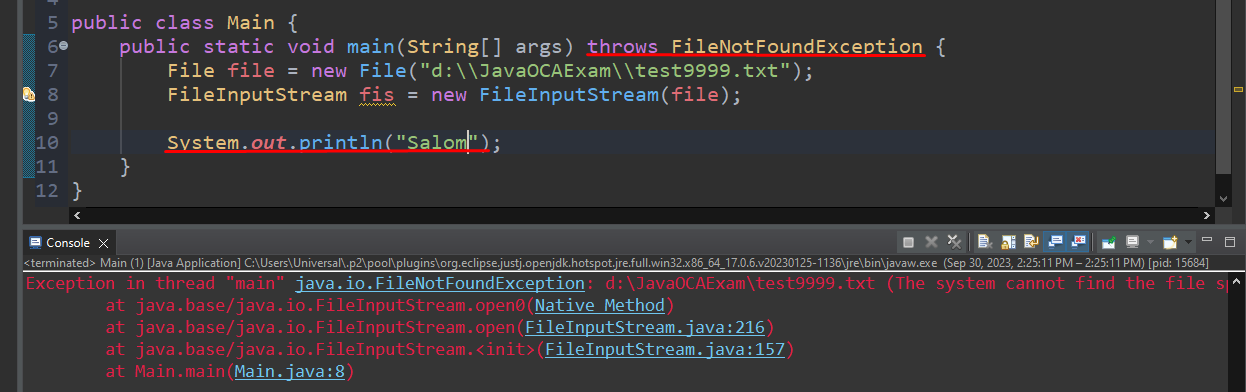
Agar istasak bizning biror methodimiz runtime exception da exception otib qolishi mumkin deb, shu exceptionni e’lon qilib qo’yishimiz mumkin. Pastda xuddi shunday qilib 3-qatorda foo() methodda exceptionni e’lon qilib qo’yganmiz:



Endi 2-tur exception **Checked** exception haqida gaplashamiz. Checked exception da biz har doim majburmiz kelajakda otilishi mumkin bo’lgan exception ni yo try-catch blockda qayta ishlashga yo biror methodda signature sifatida o’sha exceptionni e’lon qilishga. Masalan pastdagi misolda checked exception da farqi yo’q **test9999.txt** faylimiz bormi yoki yo’qmi yoki faylimizni muvofaqiyatli o’qiydimi(yozadimi) farqi yo’q. Har doim har ehtimolga qarshi exception tashlash shart. Va bu exceptionni pastdagi misol kabi try-catch da tutib qayta ishlashimiz mumkin:



Yoki bo’lmasam exception ni method signatureda e’lon qilish mumkin. Bunda kelajakda shu **main()** m.miz **FileNotFoundException** tashlab qolish ehtimoli bor deganini bildiradi:



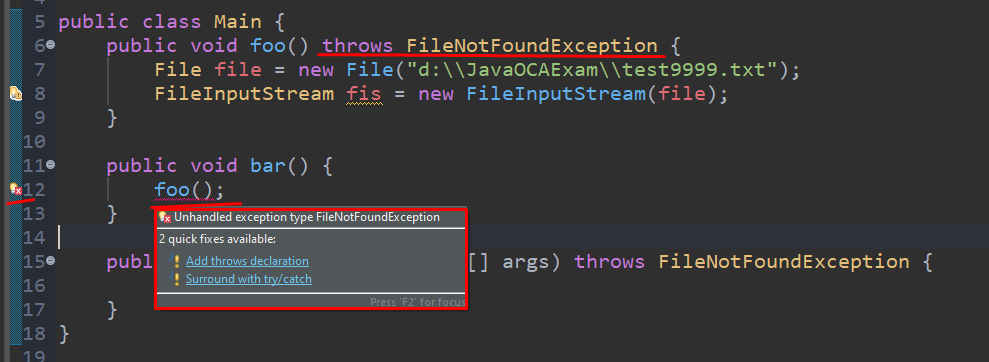
E’tibor bergan bo’lsangiz baribir console da xatolik chiqdi. Lekin bundan oldingi misolimizda try-catch ni ishlatganimizda console da xatolik yo’q edi. Asosiy farqi ham shunda, try-catch ni yozsak, exceptionimizni catch blockda qayta ishlashimiz mumkin, masalan message chiqarishimiz mumkin. Eng asosiysi xatolikdan keying kodlar ham ishlayveradi. Chunki qayta ishlaganimiz uchun. Lekin exception ni e’lon qilsak method signatureda, u holda biz bu exceptionni qayta ishlay olmaymiz. To’g’ri compile time da xatolik bermaydi, lekin run timeda bajarilish vaqtida xatolik kelib chiqadi. Yuqoridagi misolda ko’rdikki 8-qatorda xatolik chiqdi, chunki bunday file yo’q bizda, shu yerda kodimiz ishlashdan to’xtadi va 10-qatordagi sout(“Salom”) kodimiz bajarilmadi.

Demak checked exceptionlarni doim yo try-catch blockka olishimiz kerak yo method signatureda shu otilishi mumkin bo’lgan exceptionni e’lon qilishimiz kerak.

Yana bir narsa muhim. Deylik bizda foo() m. bor bo’lsin. Uni ichida exception tashlaydigan kod bor. Agar biz shu foo() m.ni ichidagi ecxeptionni try-catch blockning ichida qayta ishlasak, u holda biz bu foo() m.ni boshqa bir deylik bar() m.ni ichida bemalol chaqirib ishlata olamiz:



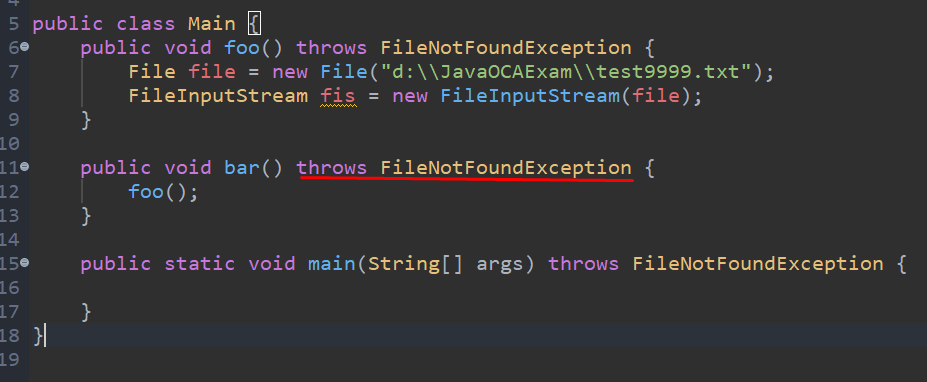
Lekin foo() m.ni signaturesida shu exceptionni e’lon qilsak, u holda biz bu methodni boshqa bir bar() m.ni ichida chaqirsak, bar() m.da xatolik sodir bo’ladi. Sababi bar() m. biladiki foo() m. try-catch bilan qayta ishlanmaganini va shu bar() m.da ham shu FileNotFoundException exception tashlanishini, Shuning uchun biz xatolik olamiz:



Xo’sh bundan qutulish uchun nima qilish kerak. Buni 2 ta usul bilan hal qilamiz. 1-usul bu bar() m.ni ichiga try-catch bilan foo() m.ni chaqirgan qismimizni try-catch bilan qayta ishlashimiz zarur:

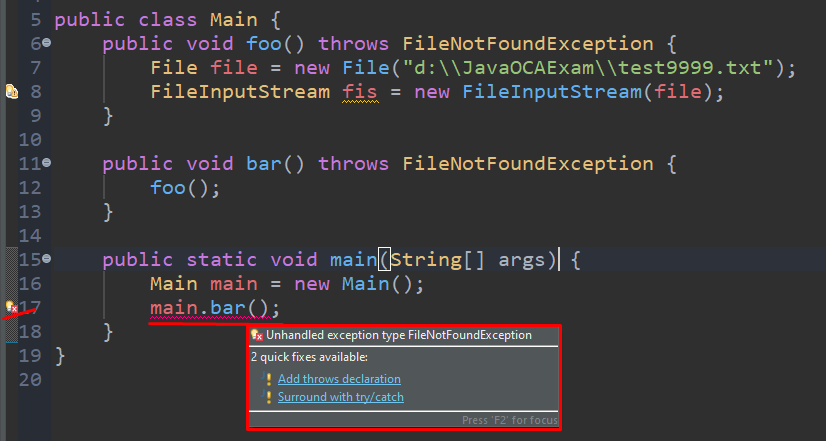


Yoki 2-usuli bu bar() m.ni signaturasiga o’sha exceptionni ham yozsih kerak:

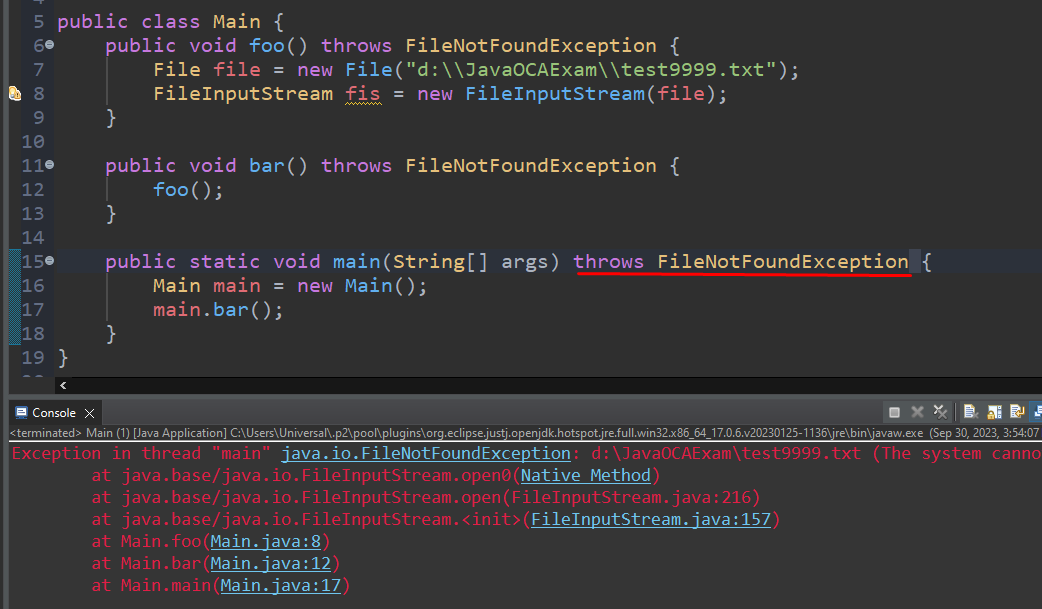


Xo’sh yuqoridagi 2 ta usulni nima farqi bor. 1-usulda agar foo() m.da exception tashlansa, uni bar() m.da try-catch bilan qayta ishlash mumkin. Lekin 2-usulda biz uni qayta ishlay olmaymiz, chunki exceptioin bar() m.ni signaturesida e’lon qilingan va uni ichida try-catch yozilmagan qayta ishlash uchun.

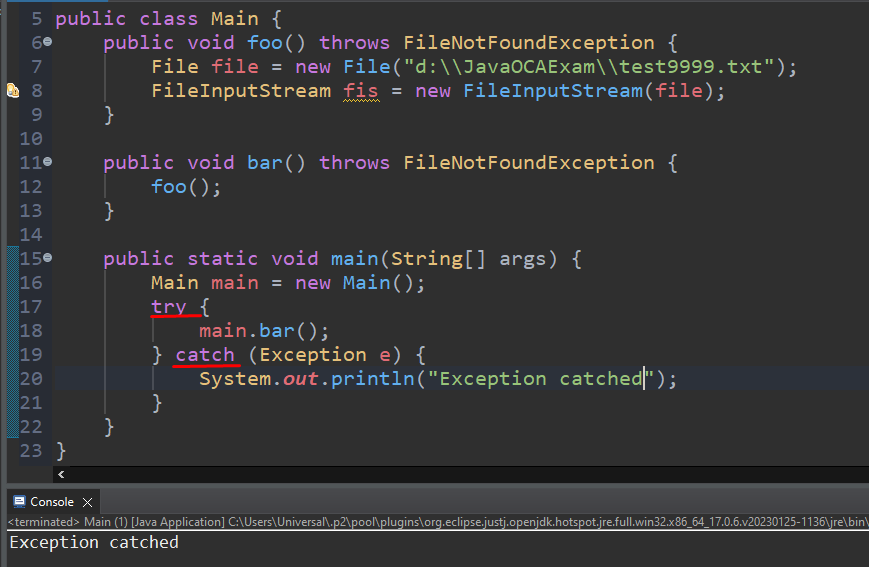
Agar biz kelajakda exception otishi aniq bo’lgan kodimizni foo() m.ni boshqa bir bar() m.da chaqirsak va bu bar() m.ni main() methodda shu classdan object olib yaratib chaqirsak, xatolik beradi, chunki exception main() methodda ham tashlanadi . Sababi biz main() m.da foo() m.ni chaqirdik:



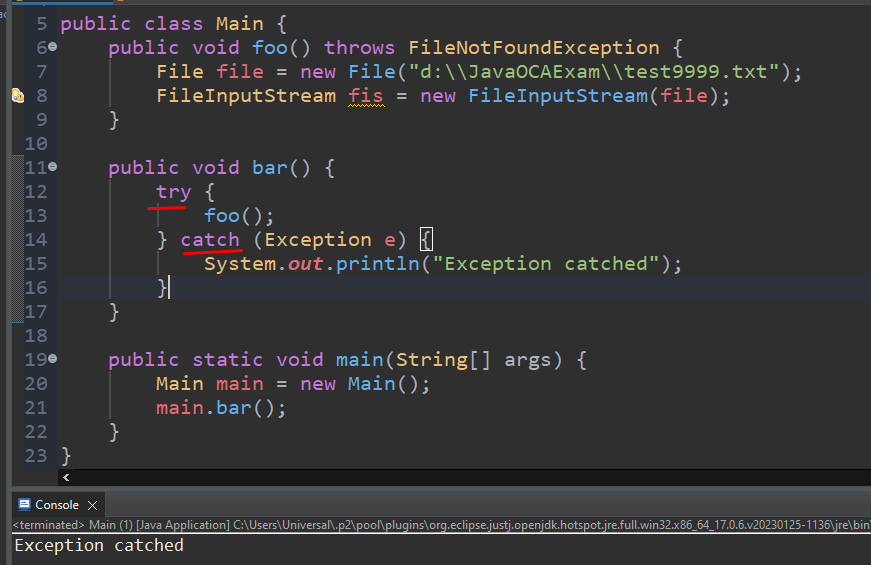
Buni yechimi avvalgidek 2 ta usuldan foydalanishdir. 1-usul bu main() m.ga signaturesiga exception yozishdir:



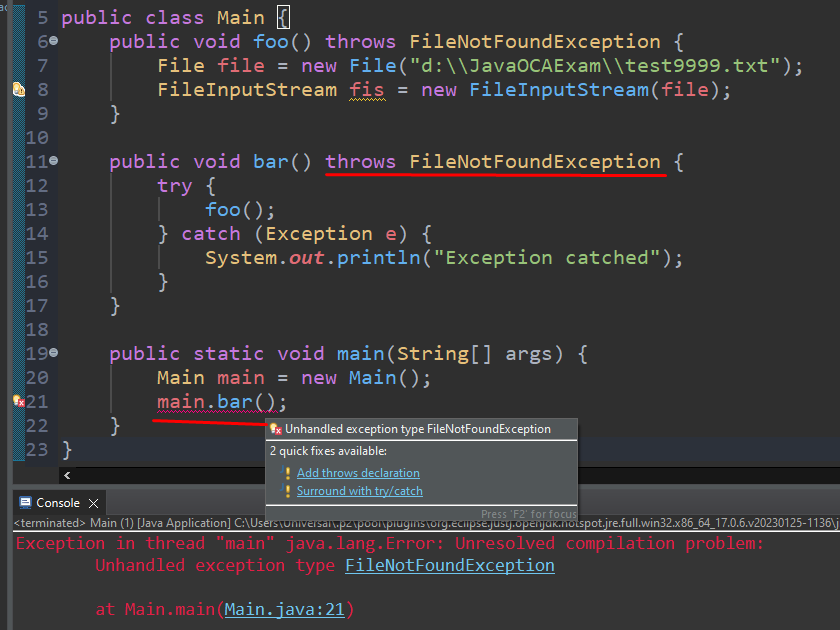
2-usuli esa main() m.da try-catch dan foydalanishdir. Bu usulda exception qayta ishlanadi:



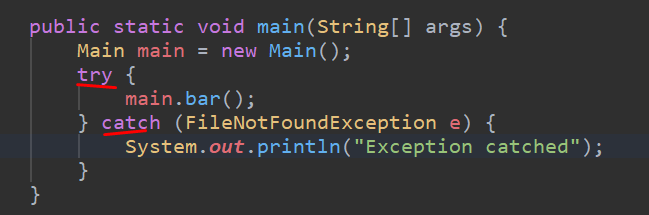
Bu 2 ta usuldan tashqari yana bitta usuli bor. Bunda yo foo() m.da yo bar() m.lardan bittasida try-catch blockni yozib, qayta ishlash kerak. Shundagina main() m.ga hech qanday exception otilmaydi. Pastda biz bar() m.da exceptionni try-catch bilan qayta ishlab oldik. Shuning uchun main() m.da hech qanday try-catch block yoki exception signatureni yozmadik:



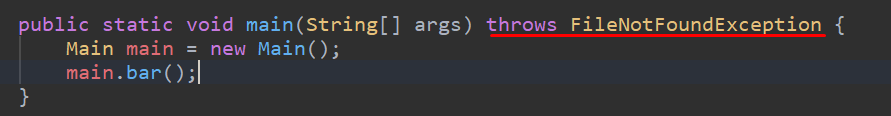
Lekin yana bir muhim joyi shundaki, agar biz pastda bar() m.da try-catch bilan exceptionni qayta ishlagan bo’lsak hamki, main() m.ga exceptioni otilyapti. Xo’sh sababi nima buni. Sababi shuki, e’tibor bergan bo’lsangiz, bar() m.ni method signaturesiga throws FileNotFoundException ni yozdik. Bu shuni anglatadiki, bar() m.da foo() m.dan exception kelsa, uni try-catch bilan qayta ishlaydi. Lekin bar() m.ni o’zi ham exception otayapti throws FileNotFoundException signature yordamida:



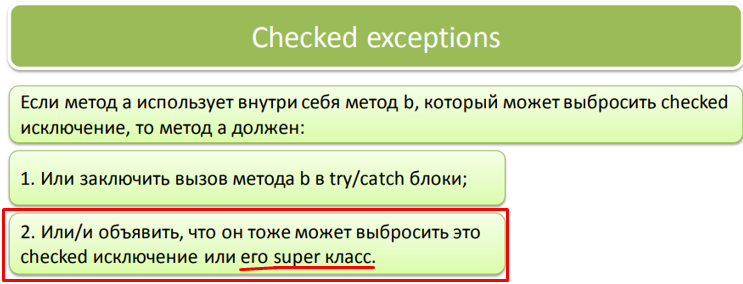
Xo’sh buni qanday yechamiz? Bunday holatda main() m.da bar() m.ni chaqirsak, albatta main() m.da bar() m.dan kelayotgan exceptionni qayta ishlashimiz kerak:



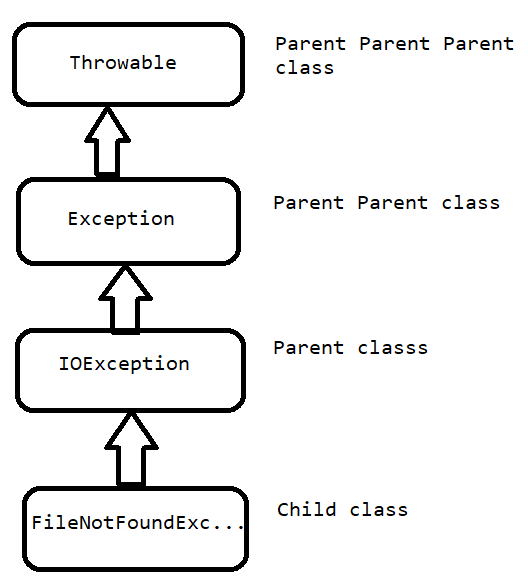
Yoki main() m.ni signaturesiga tashlanishi mumkin bo’lgan exceptioinni yozish kerak:



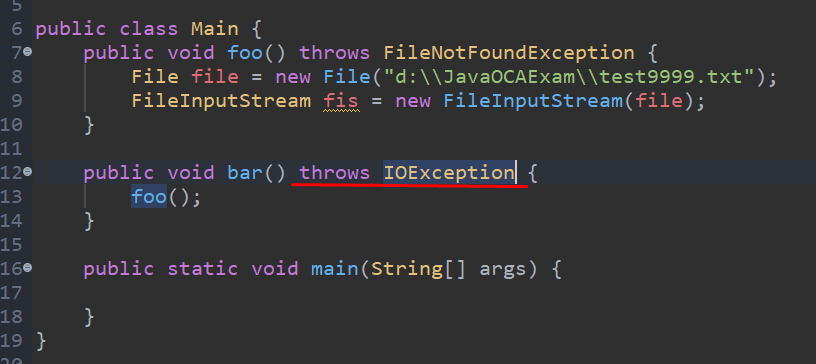
Keling pastdagi slide dan belgilangan 2-qismini ko’raylik. Agar method signature ga exception ni yozadigan bo’lsak, u holda biz bu exceptionni Parent classini yoki parent classini parent classini ham yozish mumkin:



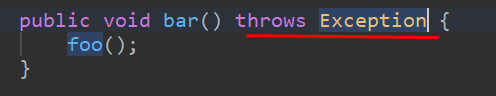
Tushunishga harakat qilaylik, masalan bizda FileNotFoundException exception bor bo’lsin, uni parent classi IOException dir, bu IOException ni parent classi esa Exception classidir, o’z navbatida Exception ni parent classi esa Throwable classidir. Pastdagi chizmadan Parent-Child classlar munosabatini ko’rish mumkin:



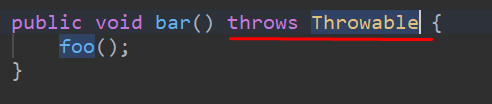
Misol ko’raylik yana oldingi misolimizni ko’ramiz. 12-qatorda ilgari FileNotFoundException ni yozgan edik. Endi uni o’rniga uni ota classi bo’lmish IOException ni yozsak ham bo’ladi:



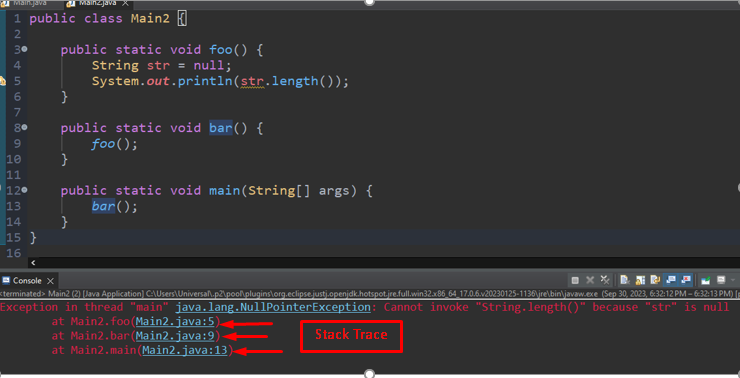
Yoki undan ham kattaroq ota classi Exception ni ham yozish mumkin:



Yoki undan ham kattaroq ota classi Throwable ni ham yozish mumkin:

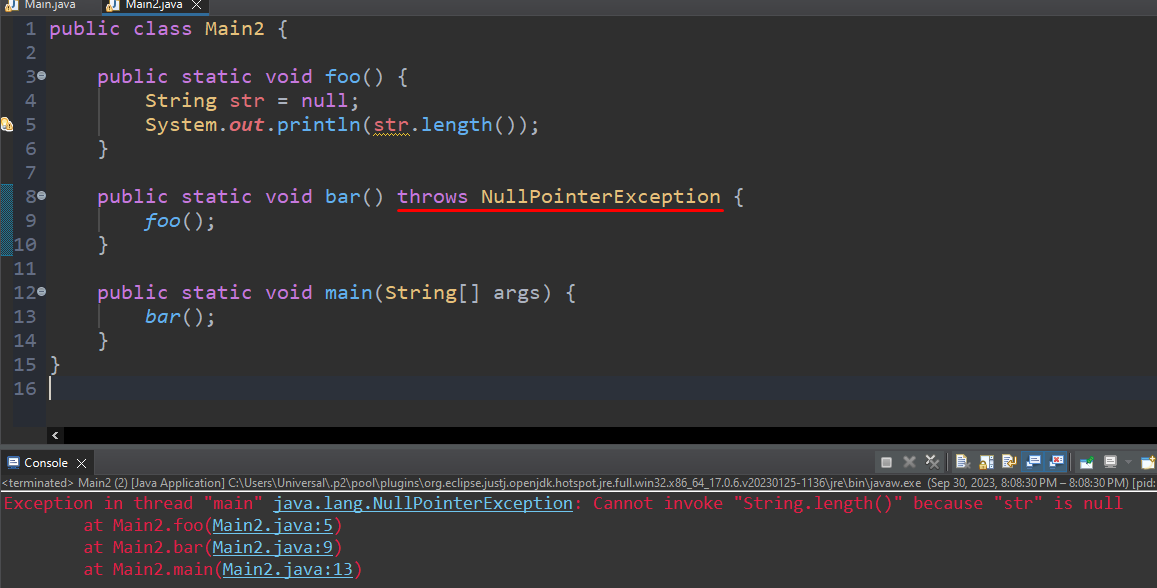


Java da Stack Trace degan tushuncha bor. Xo’sh bu nima degani? Pastdagi misolda main() m. bar() m.ni chaqiryapti, bar() m. esa foo() m.ni chaqiryapti. Shunda bizni Stack Trace imizga avval main() m. keyin bar() m. keyin esa foo() m. tushyapti. Keyin esa eng oxirgi kirgan foo() m.imiz bajarilyapti. Bajarish davomida NullPointerException exception tashlanadi. Va kodimiz bajarilmaydi. Pastdagi console da ham ko’rish mumkinki, Stack Trace ni eng pastida main() method yotibdi, keyin bar() method va eng oxirida esa oxirgi bo’lib chaqirilgan foo() m. yotibdi:



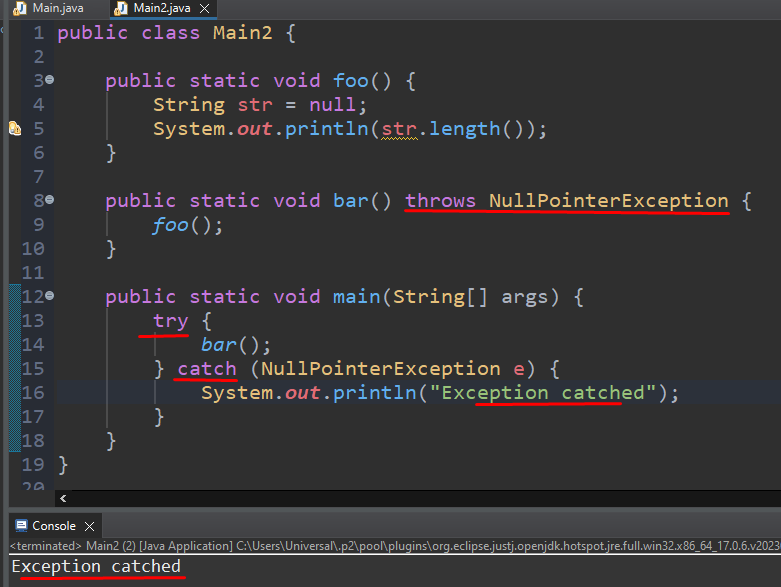
Stack Trace sezgan bo’lsangiz JS dagi Call Stack ni o’zginasi bo’lib, LIFO prinpiga asoslanib ishlaydi. Stack Trace ni eng pastida eng birinchi bo’lib chaqirilgan main() m. turibdi eng oxirida esa eng oxirgi bo’lib chaqirilgan method foo() yotibdi. Shu foo() m.ni bajarishda runtime exception hosil bo’ladi, chunki **str=null;** bo’lgani uchun. Stack Trace ning eng tepasida Exception otilgan method bo’ladi.

Runtime exception(unchecked) da biz method signaturesiga exception yozishimiz mumkin. Lekin buni yozmasak ham bo’ladi, ixtiyoriydir. Baribir runtime da exception tashlanadi:

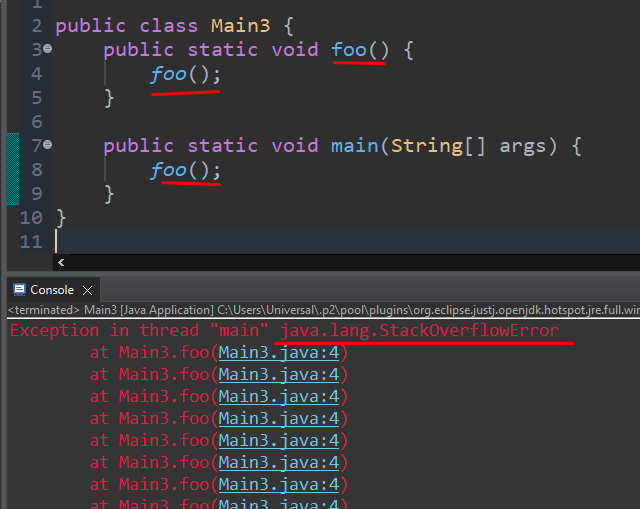


Odatda Runtime exception(unchecked) larda exception ni throw qilish yoki try-catch bilan qayta ishlash shart emas. Chunki qayerda xatolik otishi aniq va faqatgina runtimeda exception otiladi. Shuning uchun bu turdagi exceptionlarda tavsiya etilmaydi exception otish yoki try-catch bilan qayta ishlash.

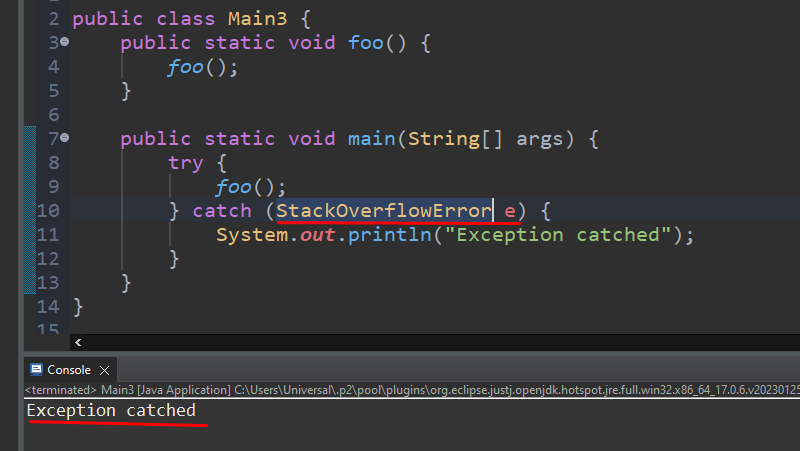
Agar istasak shu unchecked exceptionni try-catch bilan qayta ishlash mumkin. Pastdagi misolda bar() m. NullPointerException otyapti, bu exception esa main() m.da tutib olinyapti:



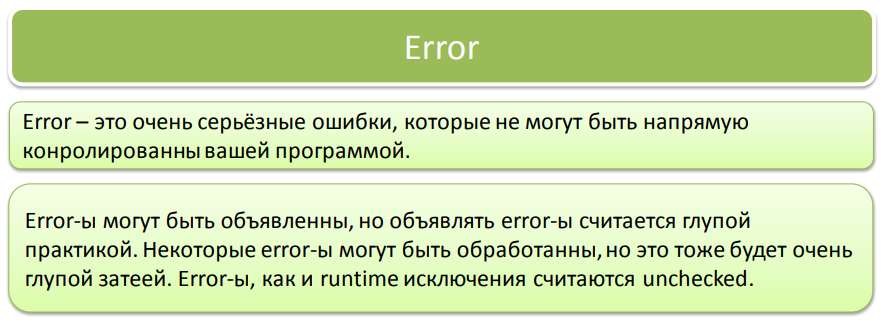
Keling endi exceptionlarni 3-turi **Error** lar haqida gaplashamiz. Error bu kodimizdagi jiddiy xatolik bo’lib, to’g’ridan-to’g’ri dasturimiz tomonidan boshqarilmaydi. Errorlarni try-catch block bilan ushlash yoki method signature exception bilan exception otish juda yomon praktika hisoblanadi, to’g’ri bu ishni qilsak xato bo’lmaydi, lekin umuman tavsiya etilmaydi. **UMUMAN ERROR LARNI TRY-CATCH BLOCKKA YOKI METHOD SIGNATURE GA OLISH KERAK EMAS!!!** Error lar unchecked(Runtime exception) hisoblanadi. Pastda recursive foo() method berilgan bo’lib, uni chaqirganimizda Stack imiz to’lib qoldi va bizga StackOverflowError tashladi:



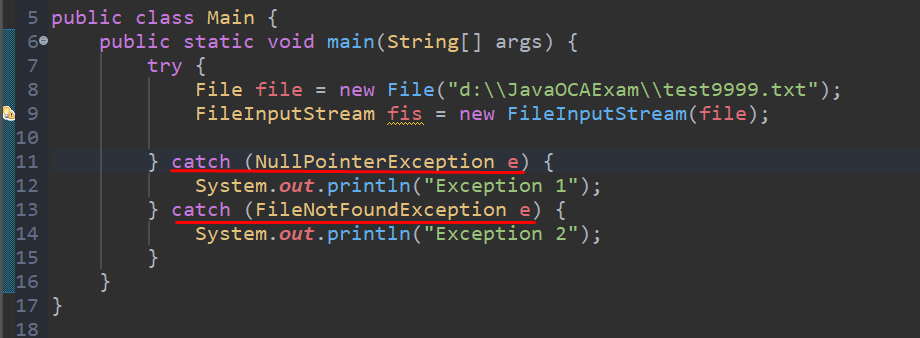
Istasak biz shu **StackOverflowError** exception ni qayta ishlashimiz mumkin **try-catch** block yordamida:



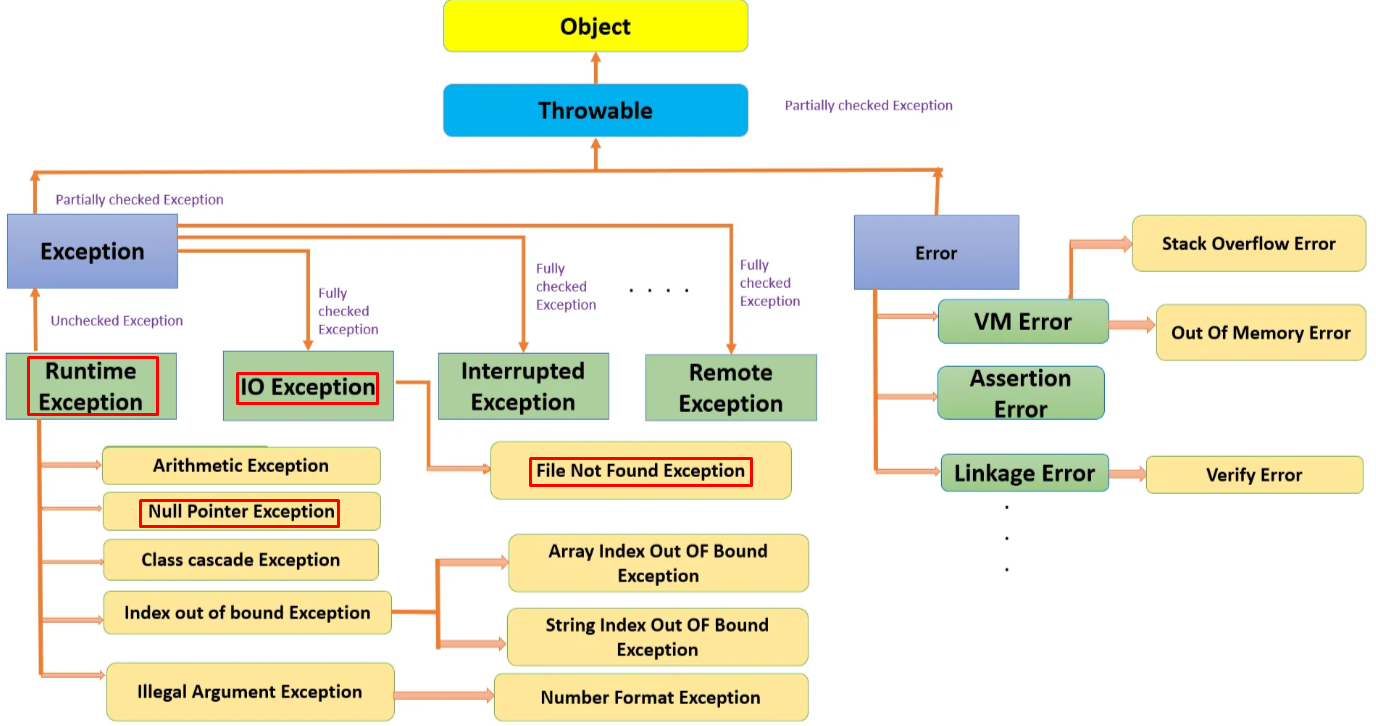
Lekin Error larni qayta ishlash umuman tavsiya etilmaydi!!!



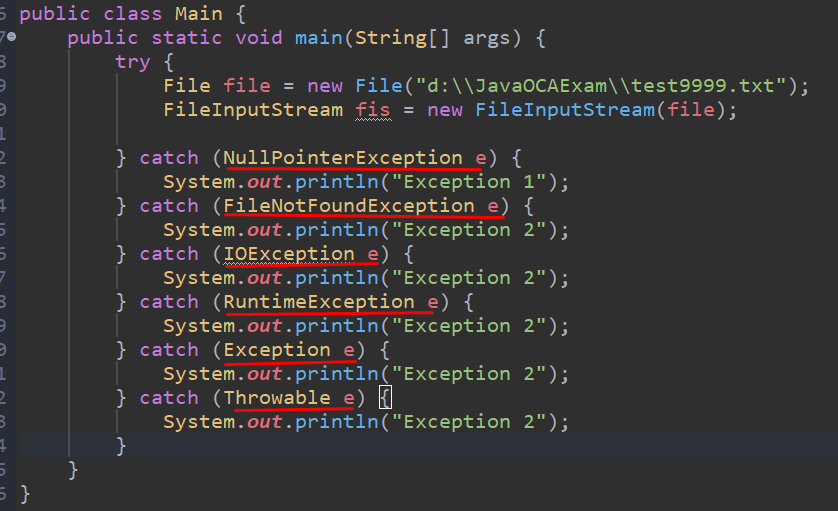
Bilamizki bizda bir nechta catch block bo’lishi mumkin. Lekin bu yerda yaxshi tushunish kerak bo’lgan holat bor. Pastdagi misol bilan ko’ramiz. Bizda 2 ta exception bor, birinchisi bu NullPointerException ikkinchisi esa FileNotFoundException. Bu 2 exceptionlar orasida hech qanday bog’liqlik, ya’ni aloqa yo’q, yana ham aniqrog’i ular exception classlarni boshqa-boshqa ierarxiyasida turibdi. Bundan tashqari ular orasida IS-A merosxo’rlik ham yo’q. Mana shunday holatlarda bu 2 ta catch blocklarni joyini almashtiramizmi yo’qmi umuman farqi yo’q, ya’ni agar FileNotFoundException tashlasa 13-qatordagi FileNotFoundException catch da tutib olinadi, agar NullPointerException chiqsa, u 11-qatordagi NullPointerException catchda tutib olinadi:



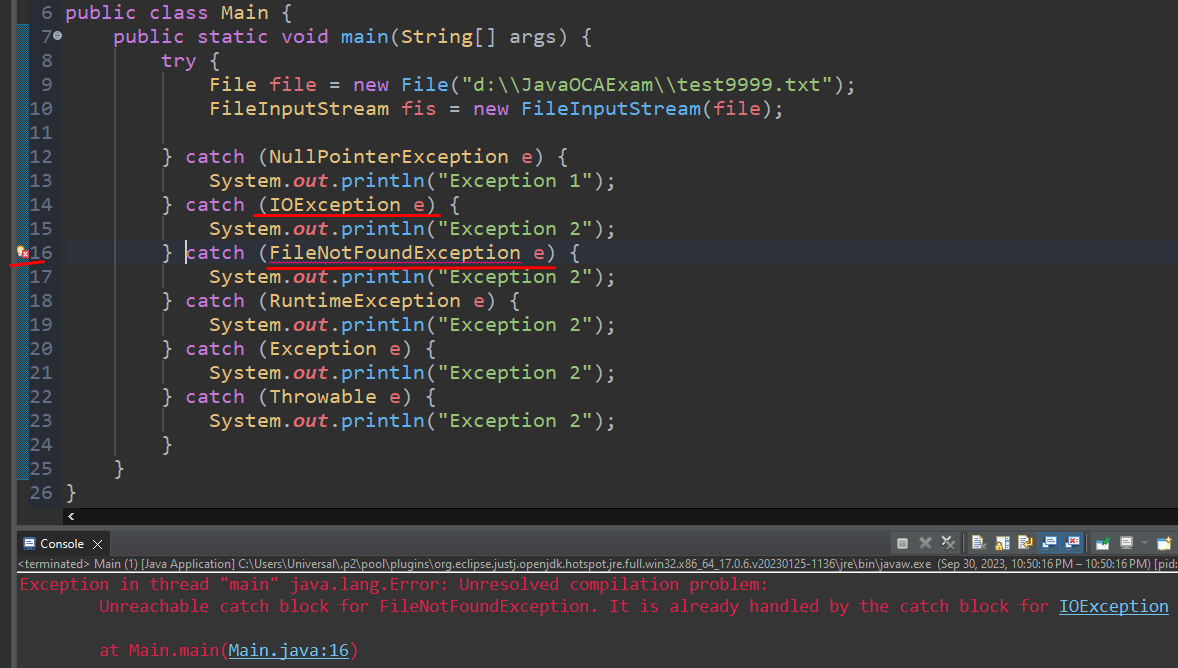
Pastdagi rasmdan ham bu 2 ta exceptionlar umuman boshqa-boshqa classlardan meros olayotganini ko’rish mumkin. NullPointerException RuntimeExceptiondan meros olyapti, FileNotFoundException esa IO Exception dan meros olyapti:



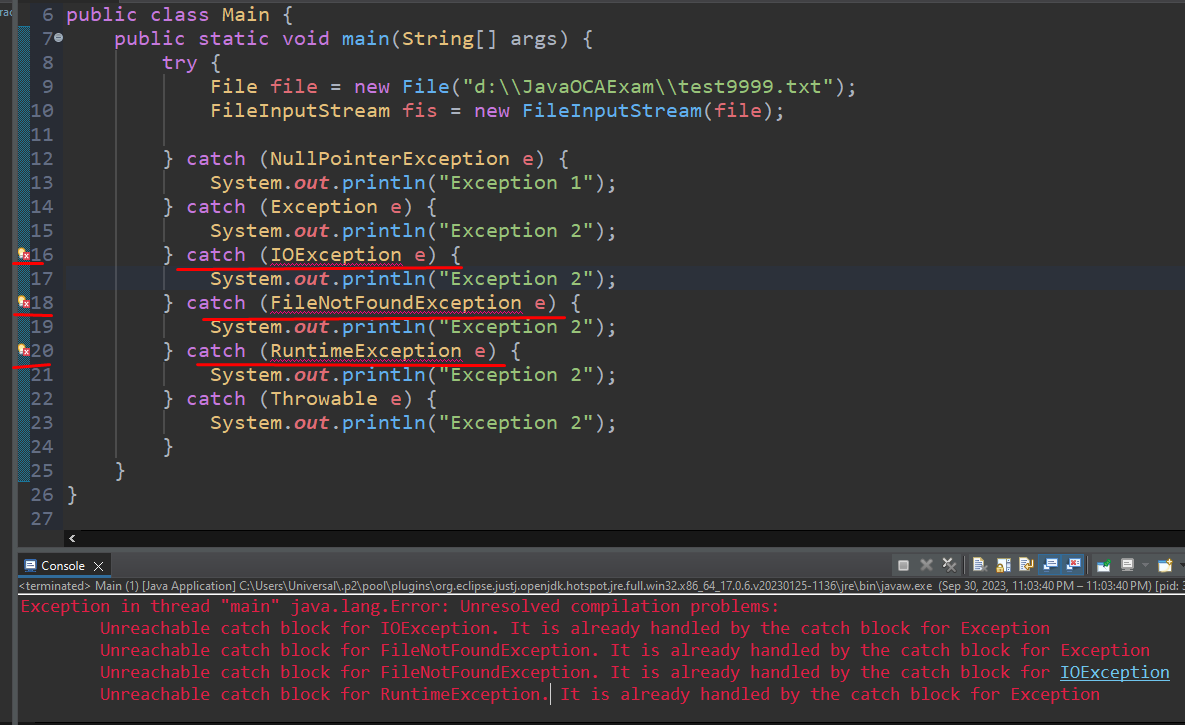
Endi agar exceptionlar o’rtasida bog’liqlik bo’lsa nima bo’ladi unda? Bunday holatda exceptionlarni yozilish tartibi juda muhim ahamiyatga ega. Pastdagi misolda o’sha tartibga amal qilib yozilgan holat aks ettirilgan:



Keling endi yuqoridagi misolda FileNotFoundException bilan IOException ni joyini o’zgartirib ko’ramiz. Pastda ularni o’rni almashgan holat berilgan. Ko’rib turibsizki, xatolik kelib chiqyapti. Xo’sh nega xatolik kelib chiqyapti. Sababi 10-qatorda FileInputStream da file FileNotFoundException otiladi. Bu exceptionni albatta NullPointerException tutib olmaydi, chunki ular umuman boshqa-boshqa parent classlardan meros oladi va ular orasida aloqa yo’q. Undan keyin 14-qatorga kelganda IOException ni ko’radi. FileNotFoundException va IOException o’rtasida bog’liqlik bormi? Ha albatta bor. FileNotFoundException classi IOException classdan meros olgan bo’lib, FileNotFoundException class child class hisoblanadi. Yuqorida berilgan chizmadan buni ko’rish mumkin. Ular o’rtasida aloqa bor ekan. Shu joyda FileNotFoundException exceptioni IOException catch blockida tutib olinadi va FileNotFoundException catch blockigacha yetib kelmaydi. Shunday holatda FileNotFoundException classi unreachable bo’ladi. Shuning uchun xatolik beradi:

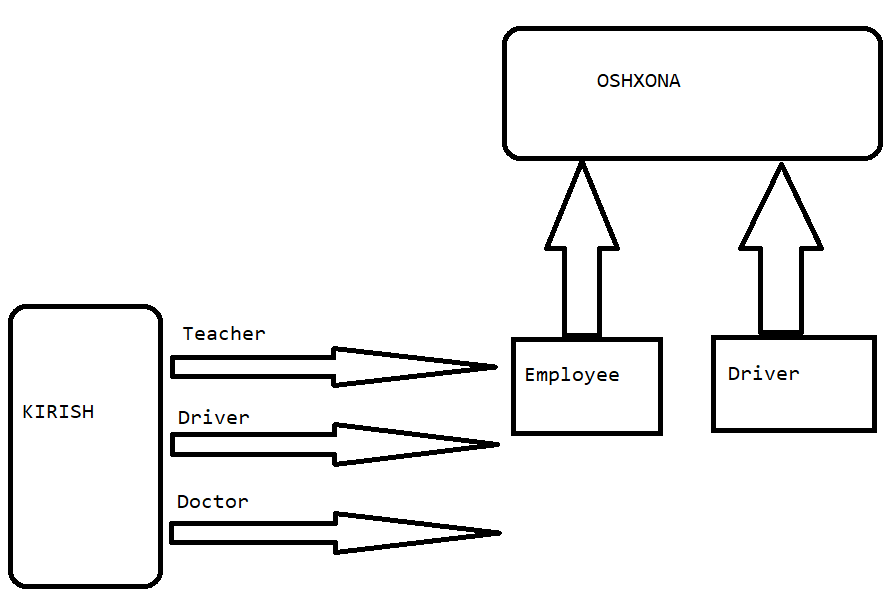


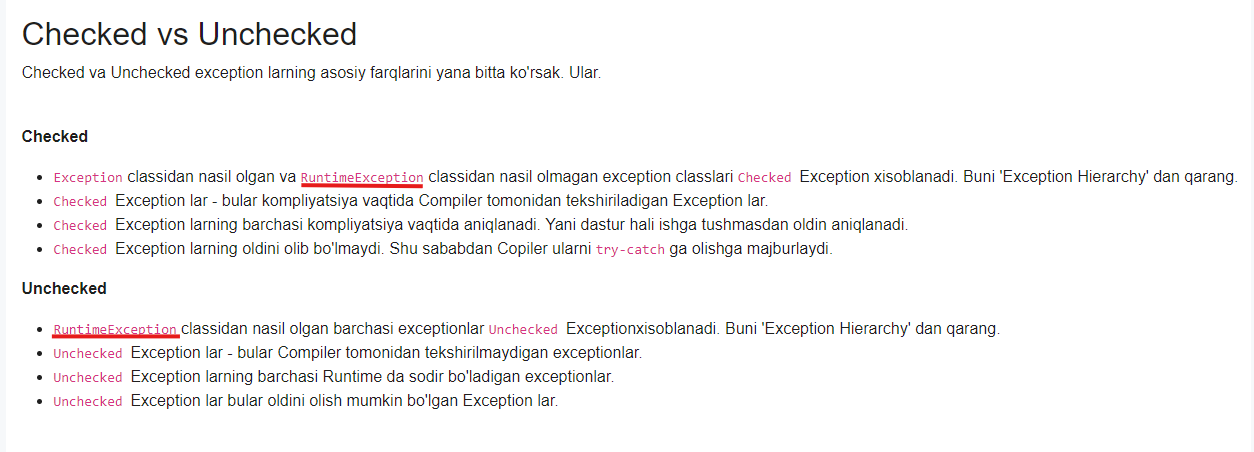
Yana misol ko’ramiz. Yuqoridagi misolni yana o’zgartiramiz. Bu safar Exception classli catch blockni yana ham yuqoriga chiqaramiz. Bu safar 3 ta catch blockda xatolik beryapti. Sababi Exception classi IOException uchun ham, FileNotFoundException uchun otasining otasi, RuntimeException uchun esa ota class hisoblanadi. Agar qanaqadir NullPointerException yoki FileNotFoundException chiqadigan bo’lsa, eng birinchi Exception class catch blockda tutib olinadi. Shuning uchun qolgan 3 ta exception catchda tutib olinmaydi, ya’ni ulargacha yetib kelmaydi va unreachable degan xatolik beradi:

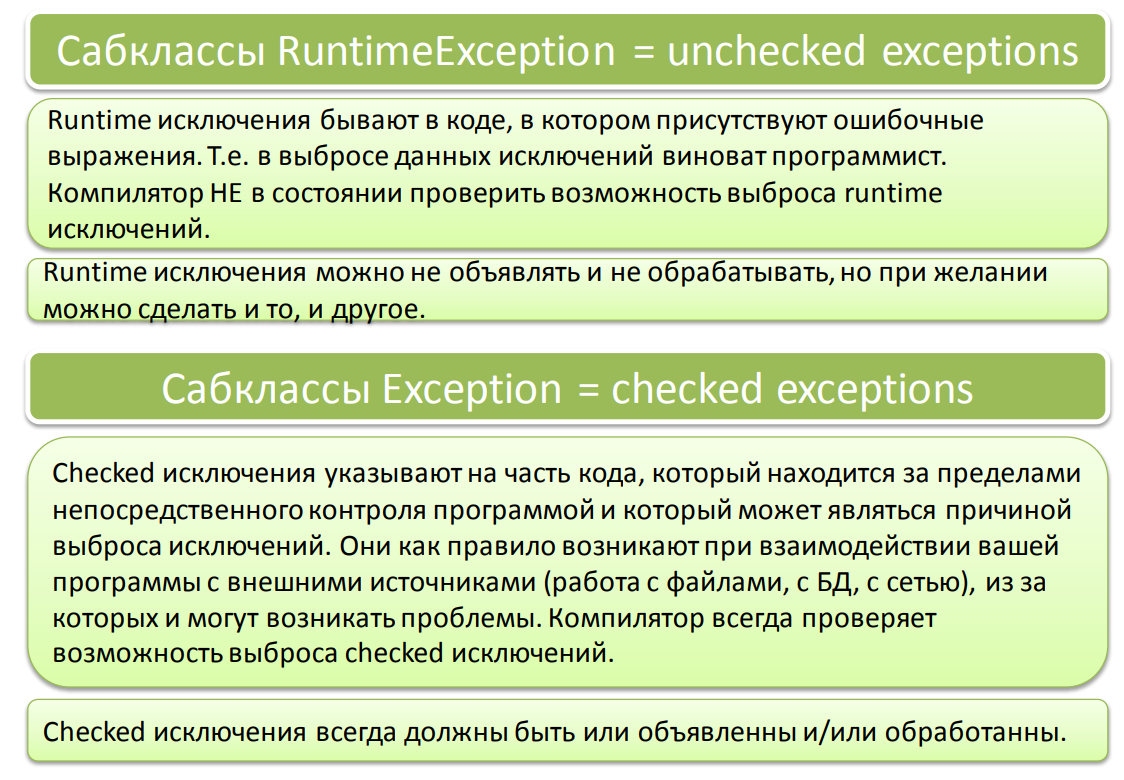


Demak, exceptionlarda shu exceptionni ota classi yoki ota classining ota classi va hokozo… birinchi yozilsa, exception ota classli exception classda tutib olinadi, keyingi bola classli exceptionga o’tmaydi. Bunda bola class unreachable bo’ladi, ya’ni exceptionni tutib olaolmaydi. Chunki ota exception classda tutib olingan bo’ladi.

Buni Zaur Tregulov juda yaxshi misol bilan tushintirgan. Masalan oshxonaga 2 ta qator bo’lib Employee va Driver lar navbatda turishibdi. Kim oshxonaga kelsa, avval tekshiriladi agar u ishchi bo’lsa Employee turgan qatorda navbatda turadi. Bizda kirishdan Teacher keladi, undan so’raladi “Sen Employee misan” deb, ha u Employee, demak Employee turgan qatorda navbatda turadi. Keyin Driver keladi undan so’raladi “Sen Employee misan” deb, ha u Employee, demak u ham Employee turgan qatorda navbatda turadi. Keyin Doctor keladi, undan so’raladi “Sen Employee misan” deb, ha u Employee, demak Employee turgan qatorda navbatda turadi. Qarabsizki hamma Employee bo’lganlar Employee da navbatda turaveradi, hech kim Driver qatorda turmaydi. Chunki bu Employee larni ota classi bu Employee dir. Xuddi shunday exceptionlarda ham Ota class exception birinchi yozilsa, bola class exception gacha bormaydi bu exception, Ota class exceptioinda tutib olib qolinadi:







Oxirgi qatorda yozilgan 

degani bu method signaturega olish kerak yoki try-catch yordamida qayta ishlash kerak deganidir.

